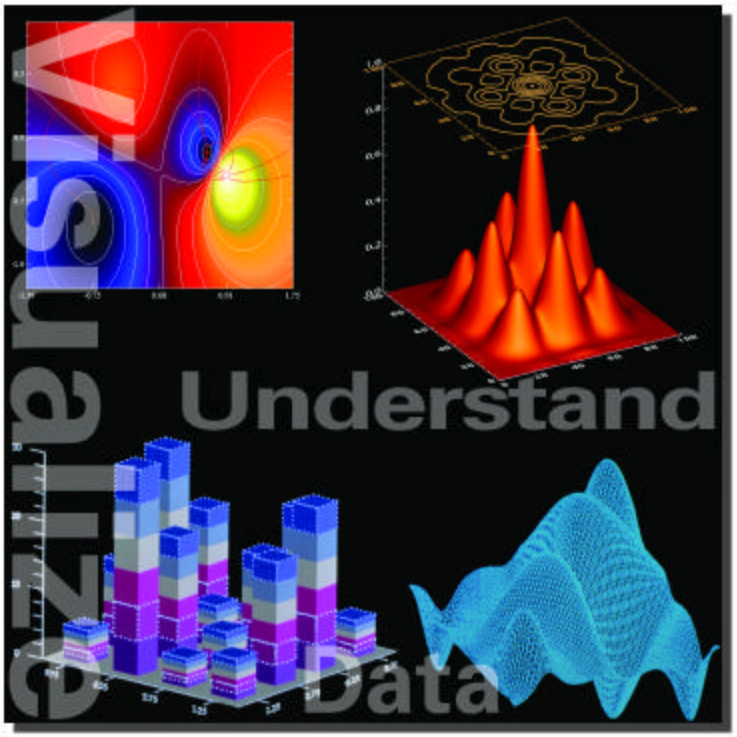
PV-WAVE 7.5®



Database Connection User's Guide

HELPING CUSTOMERS SOLVE COMPLEX PROBLEMS

Visual Numerics, Inc.

Visual Numerics, Inc.
2500 Wilcrest Drive
Suite 200
Houston, Texas 77042-2579
United States of America
713-784-3131
800-222-4675
(FAX) 713-781-9260
http://www.vni.com
e-mail: info@boulder.vni.com

Visual Numerics, Inc. (France) S.A.R.L.
Tour Europe
33 place des Corolles
Cedex 07
92049 PARIS LA DEFENSE
FRANCE
+33-1-46-93-94-20
(FAX) +33-1-46-93-94-39
e-mail: info@vni-paris.fr

Visual Numerics International, Ltd.
Suite 1
Centennial Court
East Hampstead Road
Bracknell, Berkshire
RG 12 1 YQ
UNITED KINGDOM
+01-344-458-700
(FAX) +01-344-458-748
e-mail: info@vniuk.co.uk

Visual Numerics, Inc.
7/F, #510, Sect. 5
Chung Hsiao E. Rd.
Taipei, Taiwan 110 ROC
+886-2-727-2255
(FAX) +886-2-727-6798
e-mail: info@vni.com.tw

Visual Numerics International GmbH
Zettachring 10
D-70567 Stuttgart
GERMANY
+49-711-13287-0
(FAX) +49-711-13287-99
e-mail: info@visual-numerics.de

Visual Numerics Japan, Inc.
Gobancho Hikari Building, 4th Floor
14 Gobancho
Chiyoda-Ku, Tokyo, 102
JAPAN
+81-3-5211-7760
(FAX) +81-3-5211-7769
e-mail: vda-sprt@vnij.co.jp

VIsual Numerics S.A. de C.V.
Cerrada de Berna 3, Tercer Piso
Col. Juarez
Mexico, D.F. C.P. 06600
Mexico

Visual Numerics, Inc., Korea
Rm. 801, Hanshin Bldg.
136-1, Mapo-dong, Mapo-gu
Seoul 121-050
Korea

Table of Contents

Intended Audience iii

Typographical Conventions iv

Technical Support iv

FAX and E-mail Inquiries v
Electronic Services vi

Chapter 1: Importing from a Database 1

Introduction 1

Use the SQL Syntax You Already Know 1
Import Any Data Type 2
Database Access is Convenient 2
What You Need to Know to Use these Functions 2

Connecting to a Database 2

Database Connection Example 2

Querying the Database 3

Example 1: Importing an Entire Table 4
Example 2: Importing and Sorting Part of a Table 5
Example 3: Importing and Sorting Table Summary Data 6
Example 4: Importing Data from Multiple Tables 8
Example 5: Importing NULL Values 10
Connecting to a Database from a PV-WAVE Routine 13

Controlling Rowset Size 14

Chapter 2: Reference 15

Summary of Database Routines 15

DB_CONNECT Function 16

Usage 16
Input Parameters 16
Returned Value 16
Keywords 16
Discussion 16

Example 1 17
Example 2 17
Example 3 17
Example 4 18
See Also 18

DB_GET_BINARY Function 19

Usage 19
Input Parameters 19
Returned Value 19
Keywords 19
Discussion 19

DB_SQL Function 20

Usage 20
Input Parameters 20
Returned Value 20
Keywords 20
Discussion 20
Example 1 21
Example 2 21
Example 3 21
Example 4 21
Example 5 22
See Also 22

DB_DISCONNECT Procedure 23

Usage 23
Input Parameters 23
Keywords 23
Discussion 23
Example 23
See Also 23

NULL_PROCESSOR Function 24

Usage 24
Input Parameters 24
Keywords 24
Discussion 24

Preface

This guide explains how to use the PV-WAVE:Database Connection 5.1 functions. These functions let you query a database from within PV-WAVE and import the query results into a PV-WAVE table. This imported data can then be manipulated and displayed using other PV-WAVE functions. This manual contains the following parts:

- **Preface** — Describes the contents of this manual, describes the intended audience, lists the typographical conventions used, and explains how to obtain customer support.
- **Chapter 1: Importing from a Database** — Explains how to use the database connection functions to subset and import data into PV-WAVE from an external database.
- **Chapter 2: Reference** — An alphabetically arranged reference describing each of the database functions.

Intended Audience

The PV-WAVE:Database Connection functions are easy to use if you are familiar with the target Database Management System (such as Oracle or SYBASE) and Structured Query Language (SQL). Because imported data is placed in a PV-WAVE table, you need to be familiar with the PV-WAVE table functions. These functions include BUILD_TABLE, QUERY_TABLE, and UNIQUE. They are described in the PV-WAVE Reference.

Typographical Conventions

The following typographical conventions are used in this guide:

- PV-WAVE code examples appear `in this typeface`. For example:

```
PLOT, temp, s02, Title = 'Air Quality'
```

- PV-WAVE commands are not case sensitive. In this manual variables are shown in lowercase italics (*myvar*), function and procedure names are shown in all capitals (XYOUTS), keywords are shown in mixed case italic (*XTitle*), and system variables are shown in regular mixed case type (!Version).
- Variable names that are preceded by an exclamation point (!) denote system variables.

Technical Support

If you have problems installing, unlocking, or running your software, contact Visual Numerics Technical Support by calling:

Office Location	**Phone Number**
Corporate Headquarters Houston, Texas	713-784-3131
Boulder, Colorado	303-939-8920
France	+33-1-46-93-94-20
Germany	+49-711-13287-0
Japan	+81-3-5211-7760
Korea	+82-2-3273-2633
Mexico	+52-5-514-9730
Taiwan	+886-2-727-2255
United Kingdom	+44-1-344-458-700

Users outside the U.S., France, Germany, Japan, Korea, Mexico, Taiwan, and the U.K. can contact their local agents.

Please be prepared to provide the following information when you call for consultation during Visual Numerics business hours:

- Your license number, a six-digit number that can be found on the packing slip accompanying this order. (If you are evaluating the software, just mention that you are from an evaluation site.)
- The name and version number of the product. For example, PV-WAVE 7.0.
- The type of system on which the software is being run. For example, SPARCstation, IBM RS/6000, HP 9000 Series 700.
- The operating system and version number. For example, HP-UX 10.2 or IRIX 6.5.
- A detailed description of the problem.

FAX and E-mail Inquiries

Contact Visual Numerics Technical Support staff by sending a FAX to:

Office Location	FAX Number
Corporate Headquarters	713-781-9260
Boulder, Colorado	303-245-5301
France	+33-1-46-93-94-39
Germany	+49-711-13287-99
Japan	+81-3-5211-7769
Korea	+82-2-3273-2634
Mexico	+52-5-514-4873
Taiwan	+886-2-727-6798
United Kingdom	+44-1-344-458-748

or by sending E-mail to:

Office Location	E-mail Address
Boulder, Colorado	support@boulder.vni.com
France	support@vni-paris.fr
Germany	support@visual-numerics.de
Japan	vda-sprt@vnij.co.jp
Korea	support@vni.co.kr
Taiwan	support@vni.com.tw
United Kingdom	support@vniuk.co.uk

Electronic Services

Service	Address
General e-mail	info@boulder.vni.com
Support e-mail	support@boulder.vni.com
World Wide Web	http://www.vni.com
Anonymous FTP	ftp.boulder.vni.com
FTP Using URL	ftp://ftp.boulder.vni.com/VNI/
PV-WAVE Mailing List:	Majordomo@boulder.vni.com
To subscribe include:	subscribe pv-wave YourEmailAddress
To post messages	pv-wave@boulder.vni.com

CHAPTER 1

Importing from a Database

Introduction

PV-WAVE's powerful database connection functions let you import data from an external database into PV-WAVE Advantage and CL.

Once the data is imported, you can use PV-WAVE to analyze, manipulate, and visualize the data. The database connection functions include:

- DB_CONNECT — Connect to an external database.
- DB_GET_BINARY — Connect to an external database.
- DB_SQL — Query the database with SQL SELECT statements and import the results into a PV-WAVE table.
- DB_DISCONNECT — Disconnect from the database.
- NULL_PROCESSOR — Facilitates the use of the Null_Info keyword for the DB_SQL function by extracting the list of rows containing missing for one or more columns.

NOTE Currently, the Oracle and SYBASE databases are supported. Additional databases may be supported in the future.

Use the SQL Syntax You Already Know

You can query your database from PV-WAVE using the standard SQL statements of the Database Management System (DBMS) that you are accessing. You do not need to learn new SQL syntax.

Import Any Data Type

You can import any data type that is supported by PV-WAVE. Date data is automatically converted to PV-WAVE's date/time format.

Database Access is Convenient

If you can access the database from the workstation on which PV-WAVE is running, you can connect to the database from within PV-WAVE.

If you have trouble connecting to a database from PV-WAVE, contact your database administrator.

What You Need to Know to Use these Functions

The database connection functions are easy to use if you are familiar with the target DBMS and Structured Query Language. Because imported data is placed in a PV-WAVE table structure, you need to be familiar with the functions used to manipulate tables in PV-WAVE. These functions are described in the PV-WAVE User's Guide.

Connecting to a Database

Use the DB_CONNECT function to establish a connection between a database and PV-WAVE. DB_CONNECT takes two string parameters: the name of the DBMS vendor (`ORACLE` or `SYBASE`), and the *connect_string* (a string containing login commands) for the desired database. The return value of DB_CONNECT is an identifier that is used by PV-WAVE to distinguish between database connections.

result = DB_CONNECT('*dbms_vendor*', '*connect_string*')

NOTE PV-WAVE only supports one active DBMS connection per DBMS vendor. Thus, a maximum of one Oracle and one SYBASE connection can be open at a given time. To extract data from more than one Oracle or SYBASE database, the user must disconnect from the first database before connecting to the second one.

Database Connection Example

Assume that you would like to access the data in an Oracle database. To import the data into PV-WAVE, you must first establish a connection using the

DB_CONNECT function. For example, to connect as user “scott” (password “tiger”), you could enter the following command:

```
oracle_id = DB_CONNECT(”ORACLE”, ”scott/tiger”)
```

This command attempts to connect to the default Oracle database for the current session. The default database is determined by the environment variable `ORACLE_SID`. If you want to connect to a database other than the default, you can use the following command:

```
oracle_id = DB_CONNECT(”ORACLE”, ”scott/tiger@another_db”)
```

In this case, `another_db` is the Oracle name of a database, as defined in the `TNSNAMES.ORA` file on your system.

On Sybase systems, the following syntax is supported for DB_CONNECT:

```
sybase_id = DB_CONNECT(”SYBASE”, ”scott/tiger@server:another_db”)
```

In this case, `server` is the name of the DBMS server, and `another_db` is the name of a database maintained by that server. The database name is the same as the one which is specified for the Sybase SQL command USE.

TIP Once you have imported your data into PV-WAVE, it is not necessary to maintain an open database connection. We recommend closing the connection as soon as possible, to minimize the impact on the DBMS server.

For more information on DB_CONNECT, see Chapter 2, *Reference.*

Querying the Database

After a database connection is established, you can use the DB_SQL function to issue any single-line SQL command to the DMBS. DB_SQL takes two parameters: the DBMS ID (returned by DB_CONNECT) and a string containing the SQL command. The syntax is as follows:

result = DB_SQL(*dbms_id*, "*sql_command*")

If *sql_command* returns a result set (as in a SELECT statement), *result* contains the result set, placed in a PV-WAVE table variable. In the cases where *sql_command* does not return a result set (as in INSERT, UPDATE, or DELETE statements), *result* contains a long value that indicates the success (*result*=0) or failure (*result*=–1) status of *sql_command*. The variable *result* can be manipulated and/or displayed by any PV-WAVE routine. This includes creating PV-WAVE tables which are subsets of the result set (with QUERY_TABLE, for example).

NOTE PV-WAVE single-line SQL command support does not include the ability to execute Block SQL statements. Execution of stored procedures, however, is supported, so we recommend that users who wish to perform more complicated DBMS operations from PV-WAVE enclose them in a DBMS stored procedure. For more info on creating stored procedures, contact your database administrator.

Example 1: Importing an Entire Table

The DB_SQL command shown below imports all of the data from the table called `wave.wave_prop_trx` in the Oracle database `mydbserv`. The table contains eight columns and 10000 rows.

```
oracle_id = DB_CONNECT( "ORACLE", "scott/tiger@mydbserv")
      ; Connect to the Oracle database 'mydbserv',
      ; with username 'scott' and password 'tiger'


table = DB_SQL( oracle_id, "SELECT * FROM wave.wave_prop_trx")


info, table
TABLE           STRUCT    = -> TABLE_185543239028424495098412
   Array(10000)


info, table, /Structure
** Structure TABLE_185543239028424495098412, 8 tags, 72 length:
   TRX_ID       LONG                 0
   PROP_TYPE    STRING    'OTHER               '
   PROP_ADDRESS STRING    ''
   PROP_POST_CD STRING    ''
   PROP_XGRID   DOUBLE        0.0075200000
   PROP_YGRID   DOUBLE           1.6357100
   TRX_AMT      DOUBLE           116383.00
   TRX_DATE     STRUCT    -> !DT Array(1)
```

As you can see, the data has been imported into an array of PV-WAVE structures. The tag names in the structure correspond to the column names in the database table.

Example 2: Importing and Sorting Part of a Table

In this example, we wish to import and sort a subset of the data in `wave.wave_prop_trx`. The following set of commands limits both the number of rows and columns returned to PV-WAVE.

```
oracle_id = DB_CONNECT( "ORACLE", "scott/tiger@mydbserv")
    ; Create the SQL command as a PV-WAVE variable
      ; First, create the column list
sql_command = "SELECT trx_id, prop_type, " + $
                  "trx_amt, trx_date "     + $
                  "FROM wave.wave_prop_trx "

      ; Next, add a WHERE clause to limit the number of rows
      ; This limits the subset to all dates between June 6, 1999
      ; and June 6, 2001
sql_command = sql_command + $
   "WHERE trx_date <= TO_DATE('2001/06/01', 'YYYY/MM/DD') " + $
   "  AND trx_date >  TO_DATE('1999/06/01', 'YYYY/MM/DD') "

      ; Finally add an ORDER BY clause to sort the dates in order
sql_command = sql_command + "ORDER BY trx_date"


sub_table = DB_SQL( oracle_id, sql_command)


INFO, sub_table
SUB_TABLE       STRUCT    = -> TABLE_208042343925655187313950 1
   Array(947)
INFO, sub_table, /Structure
** Structure TABLE_208042343925655187313950 1, 4 tags, 48 length:
   TRX_ID    LONG              7514
   PROP_TYPE STRING    'OTHER               '
   TRX_AMT   DOUBLE           206871.00
   TRX_DATE  STRUCT    -> !DT Array(1)


DT_TO_STR, sub_table(0).trx_date, tmp_date, tmp_time, Date_Fmt=5,
   Time_Fmt=-1
```

```
PRINT, tmp_date + ” ” + tmp_time
1999/06/01 22:20:37.000
```

TIP Very long SQL statements may not fit in a single PV-WAVE command string. For very long SQL statements, we recommend that you “build” the command in a PV-WAVE string variable, which can be any length.

Example 3: Importing and Sorting Table Summary Data

The DB_SQL command shown below imports averages by property type from table `wave.wave_prop_trx` in the Oracle database `mydb`.

```
oracle_id = DB_CONNECT( 'ORACLE', 'scott/tiger@mydbserv')

amt_by_type = DB_SQL( oracle_id, 'SELECT prop_type, ' + $
                          'AVG(trx_amt) my_avg_amt, ' + $
                          'SUM(trx_amt) my_total_amt ' + $
                          'FROM wave.wave_prop_trx ' + $
                          'GROUP by prop_type ' + $
                          'ORDER by prop_type')
    ; Select the average transaction amount
    ; for each property type, ordered by property type

INFO, amt_by_type
AMT_BY_TYPE     STRUCT    = -> TABLE_19909027124721840931719 25
   Array(9)

INFO, amt_by_type, /Structure
** Structure TABLE_19909027124721840931719 25, 3 tags, 24 length:
   PROP_TYPE    STRING    '1BR_CONDO           '
   MY_AVG_AMT   DOUBLE           80501.404
   MY_TOTAL_AMT DOUBLE           87666029.
```

NOTE When using expressions or aggregate functions in an SQL SELECT column list, we recommend that you use a column alias. This will help ensure that the tag name is valid in the PV-WAVE table variable.

This same data could also be generated with PV-WAVE functions:

```
table = DB_SQL( oracle_id, 'SELECT * from wave.wave_prop_trx')
amt_by_type_2 = QUERY_TABLE( table, 'prop_type, ' + $
                                'AVG(trx_amt) my_avg_amt, ' + $
                                'SUM(trx_amt) my_total_amt ' + $
                                'group by prop_type')
amt_by_type_2 = ORDER_BY( amt_by_type_2, 'prop_type')


INFO, amt_by_type_2
AMT_BY_TYPE_2   STRUCT    = -> TABLE_3150083162320518139151666
   Array(9)


INFO, amt_by_type_2, /Structure
** Structure TABLE_3150083162320518139151666, 3 tags, 24 length:
   PROP_TYPE    STRING    '1BR_CONDO           '
   MY_AVG_AMT   DOUBLE           80501.404
   MY_TOTAL_AMT DOUBLE           87666029.
```

TIP PV-WAVE supports some searching, sorting, and aggregate functions internally (with the WHERE and QUERY_TABLE functions, for example). In many cases, PV-WAVE searching and sorting algorithms may be faster than performing them on the DBMS server (with DB_SQL). We recommend that you try importing data into PV-WAVE with a minimum of sorting, and use PV-WAVE functions to sort, group, and search the data.

Example 4: Importing Data from Multiple Tables

This example combines data from three different tables into one PV-WAVE data set. The data is from air quality measurements from a number of fixed-location monitoring stations. One table contains the monitoring station location information (`wave.wave_ts_location`), one contains the dataset information (`wave.wave_ts_dataset`), and one contains the individual measurement data (`wave.wave_ts_datapoint`). Notice that the tag names in the PV-WAVE table variable are the same as the column alias values given in the SELECT list.

TIP We suggest that you use explicit SELECT lists (no wildcards) and column aliases when importing data through a multi-table join.

```
oracle_id = DB_CONNECT( "ORACLE", "scott/tiger@mydbserv")
    ; Create the SQL command as a PV-WAVE variable
      ; This query combines data from 3 normalized tables
sql_command = "SELECT dpnt.air_temp  air_temp, " + $
       "dpnt.humidity    humidity, " + $
       "dpnt.atm_press   atm_press, " + $
       "dpnt.o3_ppm      o3_ppm, " + $
       "dpnt.co_ppm      co_ppm, " + $
       "dpnt.no2_ppm     no2_ppm, " + $
       "dpnt.pm10_ug_m3  pm10_ug_m3, " + $
       "dset.dataset_id  dataset_id, " + $
       "dset.start_date  ref_date, " + $
       "dloc.grid_x      grid_x, " + $
       "dloc.grid_y      grid_y  " + $
"FROM wave.wave_ts_datapoint dpnt, " + $
     "wave.wave_ts_dataset   dset, " + $
     "wave.wave_ts_location  dloc "

     ; Join and data limits
     ; Only plot data for grid ID = 1
     ; And for datasets which started during 1997 through 2002
sql_command = sql_command + $
"WHERE dset.dataset_id = dpnt.dataset_id " + $
  "AND dset.start_date >= TO_DATE('19970101', 'YYYYMMDD') " + $
```

```
  "AND dset.start_date < TO_DATE('20030101', 'YYYYMMDD') " + $
  "AND dloc.loc_id = dpnt.loc_id " + $
  "AND dloc.start_date <= dset.start_date " + $
  "AND (   dloc.end_date > dset.start_date " + $
  "     OR dloc.end_date IS NULL)  " + $
  "AND dloc.grid_id = 1 "

     ; Perform the query
table = DB_SQL( oracle_id, sql_command)

INFO, table
TABLE           STRUCT    = -> TABLE_2808314677754116534184991
   Array(3400)

INFO, table, /Structure
** Structure TABLE_2808314677754116534184991, 11 tags, 72 length:
   AIR_TEMP    FLOAT           29.2000
   HUMIDITY    FLOAT           26.7000
   ATM_PRESS   FLOAT           753.520
   O3_PPM      FLOAT         0.0434300
   CO_PPM      FLOAT           3.61000
   NO2_PPM     FLOAT         0.0347400
   PM10_UG_M3  FLOAT           21.1800
   DATASET_ID  LONG                 6
   REF_DATE    STRUCT    -> !DT Array(1)
   GRID_X      FLOAT          -1.46000
   GRID_Y      FLOAT           6.15000
```

NOTE PV-WAVE only supports table JOINs during data import. JOINs are not allowed on PV-WAVE table data after import.

Example 5: Importing NULL Values

PV-WAVE does not support NULL values in table variables. If PV-WAVE encounters a NULL value in a DBMS result set, it will replace it with zero (for numeric types), a NULL string (for strings), or an empty structure (for date/time values). In the following example, we use the table `wave.wave_conv_test_nulls`, which contains the following values:

```
TEST_STRING         TEST_DATE          TEST_NUM
-----------         ---------          --------
<NULL>              04-JUL-1776        3.14
<NULL_STRING>       <NULL>             0
Not null!           04-JUL-1776        <NULL>
```

In this table, <NULL> represents the database NULL value, and <NULL_STRING> is the zero-length string (‘’). The following example indicates how this table could cause problems in PV-WAVE:

```
oracle_id = DB_CONNECT( "ORACLE", "scott/tiger@mydbserv")
table = DB_SQL(oracle_id, "SELECT * FROM wave.wave_conv_test_nulls")

INFO, table
TABLE         STRUCT    = -> TABLE_225173155029159650188791 4 Array(3)

INFO, table, /Structure
** Structure TABLE_225173155029159650188791 4, 3 tags, 48 length:
   TEST_STRING STRING    ''
   TEST_DATE   STRUCT    -> !DT Array(1)
   TEST_NUM    DOUBLE           3.1400000

INFO, table(1), /Structure
** Structure TABLE_225173155029159650188791 4, 3 tags, 48 length:
   TEST_STRING STRING    ''
   TEST_DATE   STRUCT    -> !DT Array(1)
   TEST_NUM    DOUBLE           0.0000000

INFO, table(2), /Structure
** Structure TABLE_225173155029159650188791 4, 3 tags, 48 length:
   TEST_STRING STRING    'Not null!                    '
```

```
   TEST_DATE    STRUCT    -> !DT Array(1)
   TEST_NUM     DOUBLE           0.0000000
```

In row 0 and row 1, the column `test_string` has the same value in PV-WAVE. However, in the database, the row 0 value is NULL and the row 1 value is the NULL string ‘’. Similarly, the values of `test_num` are the same in rows 1 and 2, even though they are different in the database.

If NULL-valued data is significant, one approach is to replace the NULL with a substitute value in the SELECT list. The following example indicates how this can be accomplished:

```
table_2 = DB_SQL(oracle_id, $
             "SELECT NVL(test_string, '_NULL_') test_string, " + $
                "NVL(test_date,   TO_DATE('29991231', 'YYYYMMDD')) 
   test_date, " + $
                   "NVL(test_num,    -999999.98) test_num " + $
             "FROM wave.wave_conv_test_nulls")

INFO, table_2, /Structure
** Structure TABLE_308871973212746346188263O, 3 tags, 48 length:
   TEST_STRING STRING    '_NULL_                         '
   TEST_DATE   STRUCT    -> !DT Array(1)
   TEST_NUM    DOUBLE           3.1400000

INFO, table_2(1), /Structure
** Structure TABLE_308871973212746346188263O, 3 tags, 48 length:
   TEST_STRING STRING    ''
   TEST_DATE   STRUCT    -> !DT Array(1)
   TEST_NUM    DOUBLE           0.0000000

INFO, table_2(2), /Structure
** Structure TABLE_308871973212746346188263O, 3 tags, 48 length:
   TEST_STRING STRING    'Not null!                      '
   TEST_DATE   STRUCT    -> !DT Array(1)
```

```
   TEST_NUM    DOUBLE          -999999.98
```

Another approach is the concept of *indicator variables*. An indicator variable has a value of –1 if the associated variable is NULL, and a value of zero otherwise. For an Oracle database, the following example code can be used to generate indicator variables in PV-WAVE:

```
table_3 = DB_SQL(oracle_id, $
             "SELECT test_string, " + $
              "DECODE(test_string, NULL, -1, 0) test_string_i, " + $
                 "test_date, " + $
                 "DECODE(test_date, NULL, -1, 0) test_date_i, " + $
                 "test_num, " + $
                 "DECODE(test_num, NULL, -1, 0) test_num_i " + $
             "FROM wave.wave_conv_test_nulls")

INFO, table_3, /Structure
** Structure TABLE_27395317136961263012092l7, 6 tags, 72 length:
   TEST_STRING   STRING    ''
   TEST_STRING_I DOUBLE          -1.0000000
   TEST_DATE     STRUCT    -> !DT Array(1)
   TEST_DATE_I   DOUBLE           0.0000000
   TEST_NUM      DOUBLE           3.1400000
   TEST_NUM_I    DOUBLE           0.0000000

INFO, table_3(1), /Structure
** Structure TABLE_27395317136961263012092l7, 6 tags, 72 length:
   TEST_STRING   STRING    ''
   TEST_STRING_I DOUBLE           0.0000000
   TEST_DATE     STRUCT    -> !DT Array(1)
   TEST_DATE_I   DOUBLE          -1.0000000
   TEST_NUM      DOUBLE           0.0000000
   TEST_NUM_I    DOUBLE           0.0000000
```

```
INFO, table_3(2), /Structure
** Structure TABLE_27395317136961263012092l7, 6 tags, 72 length:
   TEST_STRING   STRING    'Not null!                    '
   TEST_STRING_I DOUBLE           0.0000000
   TEST_DATE     STRUCT    -> !DT Array(1)
   TEST_DATE_I   DOUBLE           0.0000000
   TEST_NUM      DOUBLE           0.0000000
   TEST_NUM_I    DOUBLE          -1.0000000
```

Once the indicator variables have been created, it is a simple matter to create indices (with the WHERE function) which can be used to isolate or exclude the NULL values.

Connecting to a Database from a PV-WAVE Routine

You can place database connection functions in a PV-WAVE routine. Use the ON_IOERROR function to trap errors that occur while connecting, importing, and disconnecting from the DBMS. ON_IOERROR is described in the *PV-WAVE Reference*. The following example demonstrates this technique:

```
FUNCTION Read_Dept
;Read the employee name and department number
;from the database and return a new table.
ON_IOERROR, Bad
;Connect To DBMS
;===============
PRINT, 'DB_CONNECT:'
oracle_id=DB_CONNECT('ORACLE', 'scott/tiger')
PRINT, 'Ok'
;Import data from the database.
;==============================
PRINT, 'DB_SQL:'
table = DB_SQL(oracle_id, 'SELECT ename,' +$
   'deptno from emp')
PRINT, 'Ok'
;Disconnect from the DBMS.
;=========================
PRINT, 'DB_DISCONNECT: '
DB_DISCONNECT, oracle_id
PRINT, 'Ok'
PRINT, 'End'
```

```
RETURN, table
Bad:
   PRINT, 'Bad'
END
```

Controlling Rowset Size

You can control the rowset size for database queries. The rowset size is defined as the number of rows that the DBMS returns to the client per network transmission.

The ability to change the rowset size allows you to tune PV-WAVE:Database Connection to optimize the performance of each query.

Small rowsets:

- reduce the amount of temporary memory needed to import a large dataset generated by a query.
- reduce the number of blocked processes on networks with heavy traffic.
- increase the time required to complete a query.

Large rowsets:

- reduce the time required to complete a query.
- increase the amount of temporary memory required.
- increase the number of blocked processes.

To change the rowset size, modify the PV-WAVE system variable !Dbms_Rowset_Size. For example:

```
!Dbms_Rowset_Size = 300
```

The default value of !Dbms_Rowset_Size is 500.

All PV-WAVE:Database Connection routines check this variable before accepting data from the DBMS. During the same connection, the rowset size can be changed between one query and another. Changing the rowset size does not affect the total number of rows returned, just the number of network transactions which are required to return all of the rows produced by the query.

2

CHAPTER

Reference

This chapter describes each of the PV-WAVE:Database Connection routines.

Summary of Database Routines

The syntax for these routines is summarized below:

DB_CONNECT Function

dbms_id = DB_CONNECT('dbms', 'login')

Connects PV-WAVE to a database.

DB_GET_BINARY Function

list_var = DB_GET_BINARY(handle, sql_query)

Returns binary large objects (BLOBS) from a DBMS (database management system) server.

DB_SQL Function

table = DB_SQL(dbms_id, 'sql_stmt')

Queries the database currently connected to PV-WAVE.

DB_DISCONNECT Procedure

DB_DISCONNECT, dbms_id

Disconnects PV-WAVE from an external database.

NULL_PROCESSOR Function

table =
NULL_PROCESSOR(*null_info_object,[‘col1’,’col2’,…,’coln’],Comp=comp*)

Facilitates the use of the Null_Info keyword for the DB_SQL function by extracting the list of rows containing missing for one or more columns.

DB_CONNECT Function

Connects PV-WAVE to a database.

Usage

dbms_id = DB_CONNECT(*'dbms'*, *'login'*)

Input Parameters

dbms — A string specifying the database management system (DBMS).

login — A string containing the commands used to log in to the database. The string can contain the following elements:

- *user* — The username of a user authorized to connect to the database.
- *password* — The user's database password.
- *node_name* — The name of the workstation on which the DBMS is running. By default, this is the workstation you are logged onto. For SYBASE users, this is equivalent to the server name.
- *db_name* — The name of the database to connect to within the DBMS. By default this is the default database defined for your DBMS. For SYBASE users, this is equivalent to the database context.

See the *Discussion* section below for more information on the *login* parameter.

Returned Value

dbms_id — An ID number (handle) representing the DBMS.

Keywords

None.

Discussion

The following database management systems are supported:

- Oracle
- SYBASE

NOTE The syntax of the login string may vary slightly depending on the type of DBMS you are using. See your DBMS documentation or database administrator for additional information on the login string syntax.

To connect to a SYBASE database, the SYBASE environment variable must be set properly. See your database administrator if you have any questions about setting this environment variable.

TIP If you have trouble connecting to an Oracle database, it may be that environment variables are improperly set. If you cannot establish a connection to Oracle, first be sure that you can use sqlplus to access the database you want to connect to from the workstation on which PV-WAVE is running. If this works you may have to set the ORACLE_HOME and ORACLE_SID environment variables. See your database administrator for information on setting these environment variables properly.

Example 1

This example shows the default connection, where the DBMS is ORACLE, and the username and password are given in the login string. In this case, it is assumed that the database is running on the workstation the user is currently logged onto, and the database the user wants to access is set up as the default database for the DBMS.

```
oracle_id = DB_CONNECT('ORACLE','scott/tiger')
```

Example 2

This example shows a login string that specifies:

- the workstation (dbnode) running the DBMS and
- the database (mydb) to connect to within the DBMS.

```
oracle_id = DB_CONNECT('ORACLE', 'scott/tiger@mydb')
```

Example 3

This example shows a login string used to connect to a SYBASE database. The syntax of the login string for a SYBASE connection is:

user[/*password*][*@server*][:*database*]

Only the username parameter is required. If the server and/or dbase parameters are not specified, the default server and database are used. The interfaces file in the $SYBASE directory contains a list of available servers.

The following login string specifies:

- the database server (SYBASE) and
- the database context (pubs2)

```
sybase_id = DB_CONNECT('SYBASE', 'scott/tiger@SYBASE:pubs2')
```

Example 4

This example connects PV-WAVE to a SYBASE database. In this example, the *server* parameter is not specified because the default server is being used:

```
sybase_id = DB_CONNECT('SYBASE' 'scott/tiger@:pubs2')
```

See Also

DB_SQL, DB_DISCONNECT

See the following related functions in the *PV-WAVE Reference*:

BUILD_TABLE, GROUP_BY, ORDER_BY, QUERY_TABLE, UNIQUE

DB_GET_BINARY Function

Returns binary large objects (BLOBS) from a DBMS (database management system) server.

Usage

list_var = DB_GET_BINARY(*handle, sql_query*)

Input Parameters

handle — DBMS connection handle (returned by DB_CONNECT).

sql_query — A string containing an SQL statement to execute on the DBMS server. It must be a query (SELECT) statement.

Returned Value

list_var — A PV-WAVE LIST variable, one for each row in the query. Each element in the LIST is a PV-WAVE array of type BYTE.

Keywords

None.

Discussion

Since binary large objects (BLOBS) are transmitted from most DBMS systems in a different way from other data types, using DB_SQL to handle BLOBS would compromise performance.

For queries that return more than one row, specify the order of the rows with the ORDER BY clause in the *sql_query.*

NOTE One column will cause an error.

CAUTION The value of *sql_query* is subject to the following restrictions:

- ❑ It must be a query. UPDATE, INSERT, and/or DELETE will cause an error.
- ❑ It must only return one column. Queries that return more than one column will cause an error.

DB_SQL Function

Queries the database currently connected to PV-WAVE.

Usage

table = DB_SQL(*dbms_id*, *'sql_stmt'*)

Input Parameters

dbms_id — The DBMS ID (handle) that was returned by the DB_CONNECT function.

sql_stmt — A string containing an SQL statement used to retrieve data from the database. The SQL statement must be a SELECT statement.

Returned Value

table — A PV-WAVE table.

Keywords

Null_Info — Returns an associative array containing information on nulls in the database query result.

Discussion

This function returns a PV-WAVE table containing the requested data from the external database. You can then manipulate and visualize the imported data using any PV-WAVE functions.

All supported data types from the database can be imported into PV-WAVE variables.

Date/Time data is imported directly from the database into PV-WAVE date/time format.

PV-WAVE does not support database NULL values. NULL values are converted to zeros for numeric types, and NULL strings for type string.

You cannot retrieve image data from a SYBASE database.

The maximum length of a SYBASE data cell that can be imported into PV-WAVE is 1024 bytes.

CAUTION SYBASE Users — If you import a cell containing an undefined value, PV-WAVE will crash. Some SYBASE queries, especially ones using the `compute by` function, may produce cells containing undefined values. Usually, these undefined cells appear as blank spaces used to format the resulting table. Every data cell that you import from a SYBASE database must contain a meaningful value: e.g., an actual or NULL value.

For detailed information on working with tables in PV-WAVE, see the PV-WAVE User's Guide.

Example 1

This example imports all of the data from the `emp` table in the ORACLE database `mydb`.

```
oracle_id = DB_CONNECT('ORACLE', 'scott/tiger@Tmydb')
emp = DB_SQL(oracle_id, 'SELECT * from emp')
```

Example 2

This example imports the name, job, and salary of the managers whose salary is greater than $2800.

```
oracle_id = DB_CONNECT('ORACLE', 'scott/tiger@mydb')
emp = DB_SQL(oracle_id, "SELECT ename, job,"+$
   "sal from emp where job = 'MANAGER' and "+$
   "SAL > 2800")
```

Example 3

This example imports the names and salaries of employees whose salary is between $1200 and $1400.

```
oracle_id = DB_CONNECT('ORACLE', 'scott/tiger@mydb')
emp = DB_SQL(oracle_id, 'SELECT ename, sal'+$
   'from emp where sal between 1200 and 1400')
```

Example 4

This example imports the names of employees and their commissions whenever the commission is not a NULL value.

```
oracle_id = DB_CONNECT('ORACLE''scott/tiger@mydb')
table=DB_SQL(oracle_id, 'SELECT ename' +$
   'from emp where comm is not NULL')
```

Example 5

This example uses the *Null_Info* keyword.

```
table=db_sql(db_connect('oracle', 'scott/tiger'), 'select * from
   blanktest', null_info=foo)
```

This returns the result ‘table’ from your query and the null info object associative array ‘foo’. Foo contains three elements:

- N_ROWS = the number of rows returned in the query
- N_COLS = the number of columns or fields returned
- MISSING_DATA = the null info object associative array

The MISSING_DATA associative array contains the field name tags, each of which has the associated array listing the rows with missing data for the tag.

For more information on the null info object and to process and extract the null information array use the NULL_PROCESSOR function.

See Also

DB_CONNECT, DB_DISCONNECT, NULL_PROCESSOR

See the following related functions in the *PV-WAVE Reference*:

BUILD_TABLE, GROUP_BY, ORDER_BY, QUERY_TABLE, UNIQUE

DB_DISCONNECT Procedure

Disconnects PV-WAVE from an external database.

Usage

DB_DISCONNECT, *dbms_id*

Input Parameters

dbms_id — The DBMS ID (handle) that was returned by the DB_CONNECT function.

Keywords

None.

Discussion

Use this function when you:

- are finished importing data from a database and want to end the session and free the DBMS license seat.
- have accessed data from one database (e.g., `mydb`) and want to access data from a different database (e.g., `yourdb`).
- want to access the same database (e.g., `mydb`), but using a different login string.

Example

In this example, the DB_DISCONNECT procedure is used to disconnect from the `ORACLE` database.

```
oracle_id = DB_CONNECT('ORACLE','scott/tiger')
emp = DB_SQL(oracle_id, 'SELECT * from emp')
DB_DISCONNECT, oracle_id
INFO, /Structure, emp
```

See Also

DB_SQL, DB_CONNECT

See the following related functions in the *PV-WAVE Reference*:

BUILD_TABLE, GROUP_BY, ORDER_BY, QUERY_TABLE, UNIQUE

NULL_PROCESSOR Function

Facilitates the use of the *Null_Info* keyword for the DB_SQL function by extracting the list of rows containing missing for one or more columns.

Usage

table =
NULL_PROCESSOR(*null_info_object,['col1','col2',…,'coln'],Comp=comp*)

Input Parameters

null_info_object — The object returned by the *Null_Info* keyword in the DB_SQL call.

col_i — The list of column names.

Keywords

Comp=comp — Produces the complement to the result, that is, the result contains a list of rows with missing data. *comp* contains a list of rows with no missing data.

Discussion

Assuming the following use of the DB_SQL *Null_Info* keyword:

```
table=db_sql(db_connect('oracle', 'user_id/user_pw'), 'select *
   from blanktest', null_info=foo)
```

where `blanktest` contains the data given below, which has missing data for ID_NO in the 4th, 9th, and 11th rows and missing data for ANIMAL_NAME in the 3rd, 8th, and 10th rows.

ID_NO	ANIMAL_ NAME
1	golden
2	chirpy
3	NULL
NULL	harry
5	KC
6	skip
7	sparky
8	NULL
NULL	sneakers
10	NULL
NULL	harvey

NOTE Note: NULL indicates a NULL value in the corresponding database field.

Then,

```
jjj=NULL_PROCESSOR(foo,['ID_NO','ANIMAL_NAME'],Comp=comp)
```

produces the results

```
jjj = 2        3        7        8        9       10
comp = 0            1            4            5            6
```

This output can be utilized as in the following examples.

```
Table2 = table(comp)
```

produces a table with only rows and no missing values or as in the table given above.

ID_NO	ANIMAL_ NAME
1	golden
2	chirpy
5	KC
6	skip
7	sparky

Then,

```
Table3=table(jjj)
```

produces a table containing only rows with missing data (note how zeros have been substituted for values of ID_NO that are missing).

ID_NO	ANIMAL_ NAME
3	
0	harry
8	
0	sneakers
10	
0	harvey

Instead, if you want only the locations where one field is missing, a different db_sql call, jjj=foopro(foo,['ID_NO'],Comp=comp), returns an array, `jjj`, with the rows where ID_NO is missing (`3 8 10`).

Remember that rows are counted beginning with 0.